Tycho 2 User Guide (Draft)

Kris Garrett

June 29, 2016

1 Introduction

Tycho 2 is a reworking of a code named Tycho written by Shawn Pautz around the year 2000. The original code solved a linear, kinetic, transport equation on an unstructured, 3D tetrahedral mesh and was parallelized via MPI. The original code was made to test the performance of transport sweeps which will be subsequently described.

Tycho 2 solves the same equation with some additions. The new code adds energy dependence to the calculations to give realistic computational results for cases with several energy groups. The code is implemented in MPI and OpenMP and uses only open source software and libraries to generate meshes and compile.

2 Compiling and running the code

The code is split into the main executable named `sweep.x` and several utilities dealing with meshes in the `util` directory. All the code requires a C++11 compliant compiler. The main program, sweep.x, also requires MPI and OpenMP to compile. The utilities in the `util` directory are all serial and hence do not require MPI or OpenMP. However, depending on the utility, various libraries are needed. These will be described below.

2.1 Compiling sweep.x

Quick step guide

- Copy `make.inc.example` to `make.inc`
- Set `ASSERT` and `MPICC` in the file `make.inc`
- Type `make` to build `sweep.x`

Details To compile `sweep.x`, first copy `make.inc.example` to `make.inc` in the main project directory. The file `make.inc` contains the two user definable parts for the build process. The first user definable parameter is `ASSERT`. If `ASSERT = 1`, then the code will run much slower since bounds checking is performed on all the multidimensional arrays.

The second user definable parameter is your compiler information, which is set in `MPICC`. In this, you name the MPI compiler wrapper and any features of the compilation you want such

as optimization and debugging symbols. You must enable C++11 and OpenMP for your compiler. For an optimized compile using GCC, I usually set `mpicxx -O3 -Wall -Wextra -pedantic -std=c++11 -fopenmp`. This compile command is optimized to level 3, spits out as many warnings as possible, enables C++11 features, and enables OpenMP.

The last step is to type `make` to build the main executable. The `Makefile` will include the variables set in `make.inc`. All the object files will be stored in the `build` directory and the executable, `sweep.x`, will be created in the main project directory.

2.2 Compiling the utilities

The util directory contains several utilities for creating and partitioning meshes that sweep.x will use. When you clone Tycho 2 from github, you should get several files in the util directory.

- `cube_no_mesh.hdf` – File containing a cube's geometry. The cube is a 100x100x100 cube. This can be opened by certain CAD programs to be meshed.
- `cube-<N>.cgns` – CGNS file formatted tetrahedral meshes of `cube_no_mesh.hdf` where N is the number of cells in the mesh.
- `cube-<N>.smesh` – The same meshes as given by `cube-<N>.cgns`, just in a file format made for Tycho 2 that does not require any library dependencies to be read.
- `CreateGaussLegendre.py` – This is a Python script used to create the Gauss Legendre quadrature in the main code.
- `MakeTet.py` – This is a Python script used to make one tetrahedron in the smesh file format.
- `CreateRegularMesh.cc` – Creates a regular Cartesian mesh and then each hexahedron is split into Tetrahedra.
- `MoabToSerialMesh.cc` – Uses the MOAB mesh library to read in a common mesh format (such as CGNS, HDF5, NetCDF, etc) and convert it to the smesh file format.
- `ParallelMeshToMoab.cc` – Converts the pmesh file format into a file format that the Visit visualization tool can read.
- `PartitionColumns.cc` – Partitions an smesh into a pmesh by columns.
- `RefineSerialMesh.cc` – Refines each tetrahedron in an smesh into 8 tetrahedra.
- `SerialMeshToMoab.cc` – Converts the smesh file format into a file format that the Visit visualization tool can read.
- `SerialToParallelMesh.cc` – Partitions an smesh using the Metis partitioning library.

Creating a pmesh To run the main executable `sweep.x`, you must supply a pmesh as an argument to the executable. Several smesh files containing meshes of cubes are supplied, but it is also possible to supply your own mesh. Either the PartitionColumns or SerialToParallelMesh utility must be used to convert an smesh file into a pmesh file. PartitionColumns requires no library dependencies and SerialToParallelMesh requires the Metis partitioning library.

Creating your own mesh If you wish to supply your own mesh, you will need to convert your mesh into an smesh. This can be done using the MoabToSerialMesh utility which requires the MOAB mesh library. MOAB can read several mesh file format, but extra libraries may have to be included in the build process for your mesh format.

Visualization If you wish to visualize either an smesh or pmesh, you will use either the SerialMeshToMoab or ParallelMeshToMoab utilities. These require the MOAB mesh library, but no other dependencies are required. SerialMeshToMoab will create a .vtk file that can be read by Visit. ParallelMeshToMoab will create a .vtk file for each partition of the pmesh and a .visit file linking all the .vtk files together. You can then read the .visit file in Visit to visualize the mesh and its partitions.

The build process The build process is almost the same as for the main executable.

- Copy `make.inc.example` to `make.inc`
- Set `CPP` in the file `make.inc`
- Set any of the library directories needed by the utilities you will build
- Type `make <name of utility>`. For instance `make PartitionColumns` will create the executable `PartitionColumns.x`. Typing `make` without a target will attempt to build all the utilities.

2.3 Running sweep.x

To run `sweep.x`, type `./sweep.x <.pmesh file> <input.deck file>`. This requires that you have created a pmesh as described in the previous section and you have copied `input.deck.example` to `input.deck` (or any other name you prefer). In `input.deck`, you have the following options to set.

- `snOrder` – Order of the angular quadrature. This can be any even number from 2 to 30.
- `iterMax` – Maximum number of iterations for source iteration
- `errMax` – Tolerance for the relative error for source iteration
- `maxCellsPerStep` – Maximum number of of cell/angle pairs to compute for Ψ before communication via MPI
- `intraAngleP` – This can be 0, 1, 2, 3, or 4 for random, b-level, BFDS, DFDS, and DFHDS
- `interAngleP` – This can be 0, 1, or 2 for interleaved, globally prioritized, and locally prioritized
- `nGroups` – Number of energy group. Should be 1 or greater.
- `SweepType` Type of sweeper to use. Possible values are commented in the input.deck file.

An example `run.sh` script is supplied which will create a pmesh with 1 partition and run `sweep.x` in serial.

3 The underlying mathematics

3.1 The equation

Tycho 2 solves the following kinetic equation with isotropic scattering

$$\Omega \cdot \nabla_x \Psi(x, \Omega, E) + \sigma_t \Psi(x, \Omega, E) = \frac{\sigma_s}{4\pi} \int_{\mathbb{S}^2} \Psi(x, \Omega', E) d\Omega' + Q(x, \Omega, E). \tag{1}$$

The function Ψ is the unknown, σ_t and σ_s are the total and scattering cross sections with $\sigma_t > \sigma_s$ and are constant. The function Q is a known source.

The independent variables are:

- Space – $x \in \mathcal{D} \subset \mathbb{R}^3$,
- Direction – $\Omega \in \mathbb{S}^2$ (the unit sphere),
- Energy – $E \in \mathbb{R}^{\geq 0}$.

Notice the equation is dependent on energy, but the different energies are not coupled. This is done on purpose since the goal is to test sweeping strategies which are uncoupled in energy.

For the purpose of simpler notation, define

$$\Phi(x, E) = \int_{\mathbb{S}^2} \Psi(x, \Omega', E) d\Omega' \tag{2}$$

to get the equivalent equation

$$\Omega \cdot \nabla_x \Psi(x, \Omega, E) + \sigma_t \Psi(x, \Omega, E) = \frac{\sigma_s}{4\pi} \Phi(x, E) + Q(x, \Omega, E). \tag{3}$$

3.2 Method of discretization

Equation (3) is discretized using: discontinuous Galerkin (DG) with linear elements in x, discrete ordinates in Ω, and energy groups in E.

Energy discretization. Since equation (3) is not coupled in energy, the method of discretization of E does not matter. However, it is typical to discretize E into energy groups. Hence, the discretization in energy is denoted by E_g where g is an integer indexing the energy group (though any other discretization is fine).

$$\Omega \cdot \nabla_x \Psi_g(x, \Omega) + \sigma_t \Psi_g(x, \Omega) = \frac{\sigma_s}{4\pi} \int_{\mathbb{S}^2} \Psi_g(x, \Omega') d\Omega' + Q_g(x, \Omega). \tag{4}$$

Angle discretization. Angle is discretized via discrete ordinates, which is often denoted as S_N where N is the order of the discretization. Tycho 2 implements the Chebyshev-Legendre quadrature of the sphere. This involves a Cartesian product of N Legendre nodes on the z-axis and $2N$ equally spaced points on the circle for each Legendre node. The weights are proportional to the weights of the Legendre quadrature and scaled to sum to 4π. The nodes and weights will be denoted by Ω_q and w_q, where q stands for the quadrature index.

Equation (1) becomes

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = \frac{\sigma_s}{4\pi} \sum_{q'=0}^{2N^2-1} w_{q'} \Psi_{q'g}(x) + Q_{qg}(x). \tag{5}$$

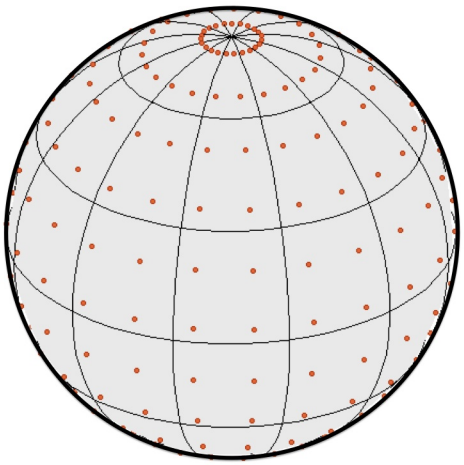
Equation (3) becomes

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = \frac{\sigma_s}{4\pi} \Phi_g(x) + Q_{qg}(x), \tag{6}$$

where

$$\Phi_g(x) = \sum_{q'=0}^{2N^2-1} w_{q'} \Psi_{q'g}(x). \tag{7}$$

Below is a depiction of the quadrature nodes on the sphere.



Spatial discretization. Since source iteration is used to converge the RHS of equation (1), it will be easier to show the equations for the simpler problem

$$\Omega_q \cdot \nabla_x \Psi_{qg}(x) + \sigma_t \Psi_{qg}(x) = Q_{qg}(x). \tag{8}$$

In this formulation, the scattering term has been absorbed into the source $Q_{qg}(x)$.

Space is discretized via linear DG. In each tetrahedral cell C_i, define basis functions $b_{ik}|_{k=0}^{3}$ to be linear interpolation functions with

$$b_{ik} = \begin{cases} 1, & \text{at vertex } k \text{ of cell } C_i \\ 0, & \text{at other vertices of cell } C_i \end{cases}. \tag{9}$$

Also, define

$$\Psi^h|_{C_i} = \sum_k \Psi_{ik} b_k. \tag{10}$$

Then the DG formulation is derived by starting with the equation

$$\Omega \cdot \nabla_x \Psi(x) + \sigma_t \Psi(x) = Q(x), \tag{11}$$

where the subscripts q, g are implicit. Then multiply by test function b_j in cell C_i and integrate to get

$$\int_{C_i} \Omega \cdot \nabla_x \Psi b_j dx + \int_{C_i} \sigma_t \Psi b_j dx = \int_{C_i} Q b_j dx. \tag{12}$$

Then use integration by parts on the first integral to get

$$-\int_{C_i} \Omega \cdot \nabla_x b_j \Psi dx + \sum_k \int_{F_{ik}} (\Omega \cdot \nu_k) b_j \hat{\Psi}^{(k)} dA + \int_{C_i} \sigma_t \Psi b_j dx = \int_{C_i} Q b_j dx. \tag{13}$$

Here, k indexes over the faces F_{ik} of cell i, and ν_k is the outward normal to face F_{ik}. The only thing left to define is the numerical flux $\hat{\Psi}^{(k)}$. Since Ψ is discontinuous at the faces, this is not well defined and must be defined by the numerical method. We use the upwind flux

$$\hat{\Psi}^{(k)} = \begin{cases} \Psi|_{F_{ik}} \text{ from } C_i, & \text{if } \Omega \cdot \nu_k > 0 \\ \Psi|_{F_{ik}} \text{ from adjacent cell}, & \text{if } \Omega \cdot \nu_k < 0 \end{cases} \tag{14}$$

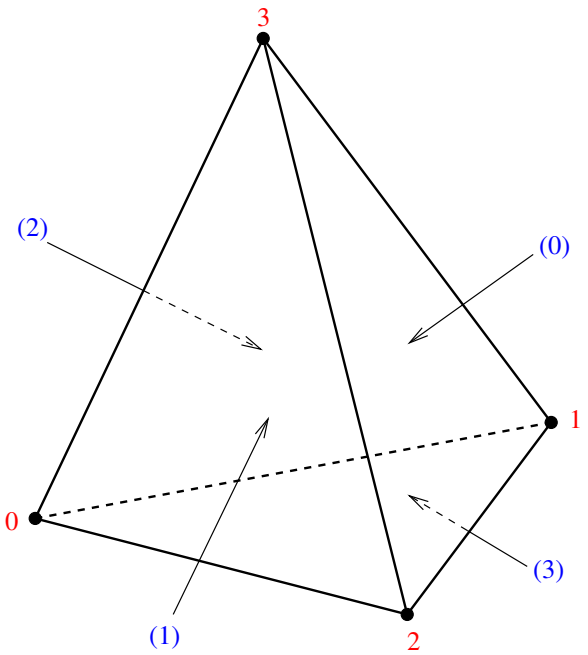
Putting everything together, one gets the linear system below. The derivation is described in an accompanying document. Face k is defined to be opposite vertex k as shown in the figure.

$$\begin{aligned} &\frac{1}{12}\bar{A}_1\left(2\hat{\Psi}_0^{(1)} + \hat{\Psi}_2^{(1)} + \hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_2\left(2\hat{\Psi}_0^{(2)} + \hat{\Psi}_1^{(2)} + \hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_3\left(2\hat{\Psi}_0^{(3)} + \hat{\Psi}_1^{(3)} + \hat{\Psi}_2^{(3)}\right) \\ &+\frac{1}{12}\bar{A}_0\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(2\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) = \frac{V}{20}\left(2Q_0 + Q_1 + Q_2 + Q_3\right) \end{aligned}$$

$$\begin{aligned} &\frac{1}{12}\bar{A}_0\left(2\hat{\Psi}_1^{(1)} + \hat{\Psi}_2^{(1)} + \hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_2\left(\hat{\Psi}_0^{(2)} + 2\hat{\Psi}_1^{(2)} + \hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_3\left(\hat{\Psi}_0^{(3)} + 2\hat{\Psi}_1^{(3)} + \hat{\Psi}_2^{(3)}\right) \\ &+\frac{1}{12}\bar{A}_1\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(\Psi_0 + 2\Psi_1 + \Psi_2 + \Psi_3\right) = \frac{V}{20}\left(Q_0 + 2Q_1 + Q_2 + Q_3\right) \end{aligned}$$

$$\begin{aligned} &\frac{1}{12}\bar{A}_0\left(\hat{\Psi}_1^{(1)} + 2\hat{\Psi}_2^{(1)} + \hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_1\left(\hat{\Psi}_0^{(2)} + 2\hat{\Psi}_2^{(2)} + \hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_3\left(\hat{\Psi}_0^{(3)} + \hat{\Psi}_1^{(3)} + 2\hat{\Psi}_2^{(3)}\right) \\ &+\frac{1}{12}\bar{A}_2\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(\Psi_0 + \Psi_1 + 2\Psi_2 + \Psi_3\right) = \frac{V}{20}\left(Q_0 + Q_1 + 2Q_2 + Q_3\right) \end{aligned}$$

$$\begin{aligned} &\frac{1}{12}\bar{A}_0\left(\hat{\Psi}_1^{(1)} + \hat{\Psi}_2^{(1)} + 2\hat{\Psi}_3^{(1)}\right) + \frac{1}{12}\bar{A}_1\left(\hat{\Psi}_0^{(2)} + \hat{\Psi}_2^{(2)} + 2\hat{\Psi}_3^{(2)}\right) + \frac{1}{12}\bar{A}_2\left(\hat{\Psi}_0^{(3)} + \hat{\Psi}_1^{(3)} + 2\hat{\Psi}_3^{(3)}\right) \\ &+\frac{1}{12}\bar{A}_3\left(\Psi_0 + \Psi_1 + \Psi_2 + \Psi_3\right) + \frac{\sigma_t V}{20}\left(\Psi_0 + \Psi_1 + \Psi_2 + 2\Psi_3\right) = \frac{V}{20}\left(Q_0 + Q_1 + Q_2 + 2Q_3\right) \end{aligned}$$



3.3 Source iteration

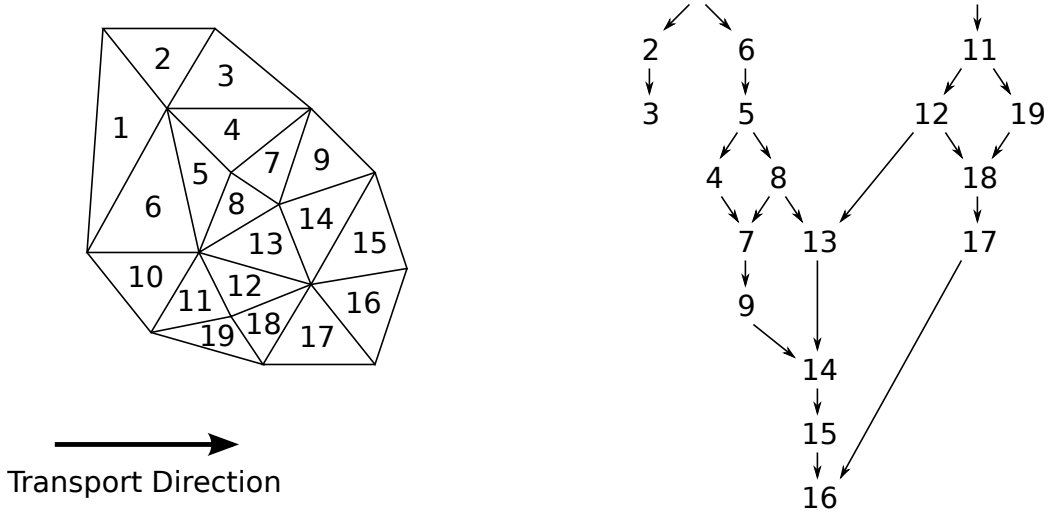
The overall algorithm is to lag the data from the integral on the RHS of equation (3). This is called source iteration

$$\Omega \cdot \nabla_x \Psi^{k+1} + \sigma_t \Psi^{k+1} = \frac{\sigma_s}{4\pi} \Phi^k + Q. \tag{15}$$

Tycho 2 iterates this process until $||\Phi^{k+1} - \Phi^k||_\infty / ||\Phi^{k+1}||_\infty <$ tolerance.

3.4 Sweeps

The most common method of solving the discretized form of equation (11) for Ψ is called a sweep. Consider the following 2D example shown below.



The sweep algorithm for the transport direction (angle) shown, shows that cells 1 and 10 have no dependencies except for the incoming boundary. Once Ψ is computed in these cells, the upwind

flux is calculated for the boundaries between (1,2), (1,6), and (10,11). With this boundary data between cells known, Ψ in cells 2, 6, and 11 can be computed. This continues until Ψ is computed in all cells.

A sweep is needed for each angle and energy group. Fortunately, all these sweeps are completely independent and therefore trivially parallelizable.